

On computational complexity of Set Automata^{*}

A. Rubtsov¹² and M. Vyalyi³¹²

¹ National Research University Higher School of Economics

² Moscow Institute of Physics and Technology

³ Dorodnicyn Computing Centre, FRC CSC RAS
{rubtsov99, vyalyi}@gmail.com

We consider a computational model which is known as set automata. A set automaton is a one-way finite automata with an additional storage—the set. There are two kind of set automata—the deterministic and the nondeterministic ones. We denote them as DSA and NSA respectively. DSA were presented by M. Kutrib, A. Malcher, M. Wendlandt in 2014 in [1] and [2]. This model seems to be interesting because it has nice properties like closure under union and intersection with a regular language and decidability of emptiness, regularity and infiniteness properties. This class looks similar to DCFL.

In this paper we show that this similarity is natural: we prove that languages recognizable by NSA form a rational cone, so as CFL. The main topic of this paper is computational complexity: we prove that languages recognizable by DSA belongs to **P** and the word membership problem is **P**-complete for DSA without ε -loops, all the languages recognizable by NSA are in **NP** and there are **NP**-complete languages. Also we prove that the emptiness problem is **PSPACE**-hard for DSA.

1 Introduction

We consider a computational model which is known as set automata. A set automaton is a one-way finite automaton with an additional storage—a set S —which is accessible through the work tape. Processing a word, a set automaton writes a query z on the work tape and performs one of the following operations: **in** is the operation of insertion word z into the set S , **out** is the operation of removing word z from the set S if S contains z , and the **test** operation is the query that verifies whether z belongs to S . After each query the content of the work tape is erased.

There are two kind of set automata—the deterministic and the nondeterministic ones. We denote them as DSA and NSA respectively.

If determinism or nondeterminism of an automaton is not significant, we use abbreviation SA, and we refer to the class of languages recognizable by (N)SA as SA. We denote as DSA the class of languages recognizable by DSA.

^{*} Supported in part by RFBR grant 17-01-00300. The study has been funded by the Russian Academic Excellence Project '5-100'.

1.1 The Definition, Known Properties and Examples

We start with a formal definition. Set automaton M is defined by a tuple

$$M = \langle S, \Sigma, \Gamma, \triangleleft, \delta, s_0, F \rangle, \text{ where}$$

- S is the finite set of states;
- Σ is the finite alphabet of the input tape;
- Γ is the finite alphabet of the work tape;
- $\triangleleft \notin \Sigma$ is the right end marker;
- $s_0 \in S$ is the initial state;
- $F \subseteq S$ is the set of accepting states;
- δ is the transition relation:

$$\delta \subseteq S \times (\Sigma \cup \{\varepsilon, \triangleleft\}) \times [S \times (\Gamma^* \cup \{\mathbf{in}, \mathbf{out}\}) \cup S \times \{\mathbf{test}\} \times S].$$

In the deterministic case δ is the function

$$\delta : S \times (\Sigma \cup \{\varepsilon, \triangleleft\}) \rightarrow [S \times (\Gamma^* \cup \{\mathbf{in}, \mathbf{out}\}) \cup S \times \{\mathbf{test}\} \times S].$$

Moreover, if $\delta(s, \varepsilon)$ is defined, then $\delta(s, a)$ is not defined for each $a \in \Sigma$.

To clarify the notation, we use \vdash notation for δ in general case and write $\delta(s, x) \vdash (s', z')$ in case $(s, x, (s', z')) \in \delta$. We define the transition relation by the action of M on configurations. A configuration is a tuple (s, v, z, \mathbb{S}) consisting of the state s , the unprocessed part of the input tape v , the content of the work tape z , and the content of the set \mathbb{S} .

$$\begin{array}{ll} (s, xv, z, \mathbb{S}) \vdash (s', v, zz', \mathbb{S}), & \text{if } \delta(s, x) \vdash (s', z'); \\ (s, xv, z, \mathbb{S}) \vdash (s', v, \varepsilon, \mathbb{S} \cup \{z\}), & \text{if } \delta(s, x) \vdash (s', \mathbf{in}); \\ (s, xv, z, \mathbb{S}) \vdash (s', v, \varepsilon, \mathbb{S} \setminus \{z\}), & \text{if } \delta(s, x) \vdash (s', \mathbf{out}); \\ (s, xv, z, \mathbb{S}) \vdash (s_+, v, \varepsilon, \mathbb{S}), & \text{if } \delta(s, x) \vdash (s_+, \mathbf{test}, s_-), z \in \mathbb{S}; \\ (s, xv, z, \mathbb{S}) \vdash (s_-, v, \varepsilon, \mathbb{S}), & \text{if } \delta(s, x) \vdash (s_+, \mathbf{test}, s_-), z \notin \mathbb{S}. \end{array}$$

We call a configuration *accepting* if the state of the configuration belongs to F and the word is processed till the end marker. So an accepting configuration has the form $(s_f, \varepsilon, z, \mathbb{S})$. A set automaton accepts word w if there exists a run from the initial configuration $(q_0, w \triangleleft, \varepsilon, \emptyset)$ to some accepting one.

Deterministic set automata were presented by M. Kutrib, A. Malcher, M. Wendlandt in 2014 in [1] and [2]. The results of these conference papers are covered by the journal paper [3], so we give references to the journal variant further.

We recall briefly their results.

Results from [3] about structural and decidability properties of DSA are presented in the following tables. In the first table we list decidability problems: emptiness, regularity, equality to a regular language and finiteness. R denotes an arbitrary regular language in all the tables. The second table describes the structural properties: L , L_1 and L_2 are languages from the corresponding class; we write $+$ in the cell if the class is closed under the operation, otherwise we write $-$.

	DSA	CFL	DCFL
$L \stackrel{?}{=} \emptyset$	+	+	+
$L \stackrel{?}{\in} \text{REG}$	+	-	+
$L \stackrel{?}{=} R$	+	-	+
$ L \stackrel{?}{<} \infty$	+	+	+

	DSA	CFL	DCFL
$L_1 \cdot L_2$	-	+	-
$L_1 \cup L_2$	-	+	-
$L_1 \cap L_2$	-	-	-
$\Sigma^* \setminus L$	+	-	+
$L \cup R$	+	+	+
$L \cap R$	+	+	+

Fig. 1. Structural and decidability properties

From Fig. 1 one can see that DSA languages looks similar to DCFL. Let us consider an example of DSA-recognizable language that is not a DCFL (and not even a CFL).

Example 1. Denote $\Sigma_k = \{0, 1, \dots, k-1\}$. We define $\text{Per}_k = \{(w\#)^n \mid w \in \Sigma_k^*, n \in \mathbb{N}\}$ to be a language of repetitions of words under Σ_k with the delimiter $\#$. For each k there exists DSA M recognizing Per_k .

Proof. Firstly M copies the letters from Σ_k on the work tape until meets $\#$ and performs **in** operation on $\#$. So after processing prefix $w\#$ of the input $\$$ contains w . Then, until reaching the right endmarker \triangleleft , M copies letters from Σ_k on the work tape and performs **test** operation on each symbol $\#$. DSA M accepts the input iff all the tests are positive: accepted input looks like $w\#w\# \dots w\# \in \text{Per}_k$.

One can naturally assume that DSA class contains DCFL (or even CFL), but this assumption is false. As was shown in [3], language $\{w\#w^R \mid w \in \Sigma^*, |\Sigma| \geq 2\}$ is not recognizable by any DSA.

Theorem 1 ([3]). *The classes DCFL and DSA are incomparable.*

Now we mention the undecidability results for NSA. These results are based on the fact that NSA can accept the set of invalid computations of a Turing machine.

Theorem 2 ([3]). *For NSA the questions of universality, equivalence with regular sets, equivalence, inclusion, and regularity are not semi-decidable. Furthermore, it is not semi-decidable whether the language accepted by some NSA belongs to DSA.*

We shall also mention a beautiful result from [3] about DSA languages under unary alphabet. It was shown that if language $L \subseteq a^*$ is recognizable by a DSA then L is a regular language. Next example shows that for NSA it is false.

Example 2. Let $\text{PRIMES} = \{a^p \mid p \text{ is a prime number}\}$. There exists NSA M with unary alphabet of the work tape recognizing language $\{a\}^* \setminus \text{PRIMES}$.

Proof. NSA M guesses a divisor k of n on the input a^n , processes a^n letter by letter and copies letters to the work tape. After first k letters M performs **in** operation, so $\mathbb{S} = \{a^k\}$. After each next k letters (the position is guessed nondeterministically) M performs **test**. M accepts a^n iff all the tests are positive and the last test was on the last letter. If M guesses k wrong at some point, then the test after this guess is negative and M doesn't accept a^n on this run. So, $a^n \in L(M)$ iff $n = km$. It is well-known that PRIMES is not a regular language and so is its compliment.

1.2 Rational transductions

Despite the classes DCFL and DSA are incomparable, their structural and decidability properties are similar. In this paper we show that this similarity is natural: we prove that the class of languages recognizable by NSA has the same structure as context-free languages.

Both classes are the principal rational cones. To define this structural property we need an auxiliary notion.

A finite state transducer is a nondeterministic finite automaton with an output tape. We give a formal definition in Section 3.2. Let T be a FST. We define set $T(u) = \{v \mid q_0 \xrightarrow[u]{v} q_f\}$ of all words v that T writes on some run from the initial to a final state while processing u . So, FST T defines a *rational transduction* $T(\cdot)$. We also define $T(L) = \bigcup_{u \in L} T(u)$.

We also use the *rational dominance* relation $A \leq_{\text{rat}} B$ which holds if there exists FST T such that $A = T(B)$.

By *rational cone* we mean a family of languages \mathbf{C} that is closed under the rational dominance relation: $A \leq_{\text{rat}} B$ and $B \in \mathbf{C}$ imply $A \in \mathbf{C}$. If there exists a language $F \in \mathbf{C}$ such that $L \leq_{\text{rat}} F$ for each $L \in \mathbf{C}$, then \mathbf{C} is a *principal* rational cone generated by F ; we denote it as $\mathbf{C} = \mathcal{T}(F)$.

Rational transductions for context-free languages were thoroughly investigated in the 70s, particularly by the French school. The main results of this research are published in J. Berstel's book [4]. As described in [4], it follows from the Chomsky-Schützenberger theorem that CFL is a principal rational cone: $\text{CFL} = \mathcal{T}(D_2)$, where D_2 is the Dyck language on two brackets.

1.3 Our contribution

One can consider D_2 as the language of correct protocols for operations with the stack (the push-down memory). In Section 3.2 we show that NSA languages

are generated by the language SA-PROT of correct protocols for operations with the set.

It was shown in [3] that the emptiness problem for DSA is decidable. In fact the proof doesn't depend on determinism of SA. We prove the lower bound for the emptiness problem for NSA and DSA: the problems are **PSPACE**-hard. In the case of unary alphabet of the work tape the emptiness problem is **NP**-hard. Our proof relies on the technique of rational cones, so we put it in Section 3.3.

The main topic of this paper is computational complexity: we prove that the word membership problem is **P**-complete for DSA without ε -loops, $\text{DSA} \subseteq \mathbf{P}$ and show that all the languages recognizable by NSA are in **NP** and there are **NP**-complete languages.

The main technical result of the paper is $\text{SA} \subseteq \mathbf{NP}$. This result is based on the fact that class SA is a rational cone and on our improvement of the technique of normal forms described in [3].

2 **P** and **NP**-complete languages

Lemma 1. *There exists a **P**-hard language recognized by a DSA.*

We describe the proof idea at first. We will reduce the language CVP (Circuit Value Problem), which is **P**-complete [5] under log-space reductions (we denote them by \leq_{\log}), to a language SA-CVP recognizable by a DSA M . The variant of the language CVP which is convenient for our purposes consists of words that are encodings of assignment sequences. An assignment has a form $P_i := P_j \text{ op } P_k$, where $\text{op} \in \{\wedge, \vee\}$, $P_i := 1$, $P_i := 0$, $P_i := \neg P_j$. An assignment sequence belongs to CVP iff the last assignment equals one.

A word from the language SA-CVP is an encoding of a sequence of reversed assignments. By reversed assignment we mean an assignment of the form $P_j \text{ op } P_k =: P_i$. Each variable P_i is encoded as a binary string, the operation basis is $\wedge, \vee, \neg, 1, 0$. In the case of operation \neg an assignment has the form $\neg P_k =: P_i$. A constant assignment has the form $c =: P_k$. Initially each variable is assigned to zero. Unlike CVP, reassignments of variables are allowed. A word w belongs to SA-CVP if the last assignment (the value of the reversed CVP-program) is equal to one. So, a word from SA-CVP looks like

$$w = \# \langle P_1 \rangle \# \wedge \# \langle P_2 \rangle \# \langle P_3 \rangle \# 1 \# \langle P_4 \rangle \# \dots \# \langle P_j \rangle \# \text{op} \# \langle P_k \rangle \# \langle P_i \rangle \#.$$

Proof. Note that the codes of the reversed assignments form a regular language. Thus we assume w.l.o.g. that DSA M processes the words, which are the sequences of reversed assignments $P_j \text{ op } P_k =: P_i$. Other words are rejected by verifying a regular event.

In the case of one-assignment $1 =: P_i$ the automaton M adds P_i to the set, in the case of zero-assignment $0 =: P_j$ the automaton removes P_i from the set. In other cases the automaton determines the value of operation by testing the variables to be members of the set. M stores the results of tests and computes the result of operation $P_j \text{ op } P_k$ in its finite memory. Then M copies the code of

P_i on the work tape and performs operation **in** if the computed result is 1 and performs **out** operation otherwise.

The automaton M also stores in its finite memory the result of the last assignment and accepts if this result is 1 when reaches the endmarker. It is clear that the automaton M accepts a reversed CVP-program iff the value of the program is 1. To complete the proof we note that the construction of a reversed CVP-program is log-space computable.

Now we show that $\text{DSA} \subseteq \mathbf{P}$.

As usually, ε -loops cause difficulties in analysis of deterministic models with ε -transitions. We say that DSA M has ε -loop if there is a chain of ε -transitions from some state q to itself.

Proposition 1. *For each DSA M there is an equivalent DSA M' without ε -loops.*

Proof. There are two kinds of ε -loops: during the first ones M only writes letters on the work tape and during the second ones M performs queries. The first ones are evidently useless, so we simply remove them. The second ones are significant: the behavior of M on these loops depends on the set's content and sometimes M goes to loop and sometimes not.

Since M is a deterministic SA, the set of words $\{u_1, \dots, u_m\}$, that M writes on all ε -paths, is finite. So we mark each state by a vector $\mathbf{a} = (a_1, \dots, a_m)$ of zeroes and ones: $a_i = 1$ iff $u_i \in \mathbb{S}$, compute all the paths on ε -loops in advance (and remove loops if they occur). Now there is no ε -loops in M' , and if s is the first state of M 's ε -loop on some run, then it turns to (s, \mathbf{a}) on the corresponding run of M' . Hereby $\delta((s, \mathbf{a}), \varepsilon) = (s', \mathbf{a}')$ and $\delta((s', \mathbf{a}'), \varepsilon) = \emptyset$.

Theorem 3. $\text{DSA} \subseteq \mathbf{P}$. *The membership problem for DSA without ε -loops is \mathbf{P} -complete.*

Proof. The input of the membership problem is an encoding of DSA M without ε -loops and w . It is easy to emulate a DSA by a Turing machine with 3 tapes.

The first tape is for the input tape of the DSA, the second one is for the work tape and the third one (the storage tape) is used to maintain the set.

Let M processes a subword u of the input between two queries to the set. During this processing the automaton writes at most $c|u|$ symbols on the work tape, where c is a constant depending only on the description of the set automaton—on each deterministic step the automaton writes only some finite word on the work tape since there is no ε -loops. So to store the set content during the emulation it is sufficient to use $c|w|$ space on the storage tape, where w is the input. Thus each query to the set can be performed in polynomial time of the input size.

So, the membership problem for DSA without ε -loops belongs to \mathbf{P} and due to Lemma 1 it is a \mathbf{P} -complete problem. By excluding ε -loops due to Proposition 1 we get that every language recognizable by a DSA belongs to \mathbf{P} .

Remark 1. In Proposition 1 we gave an exponential upper bound on the number of M 's steps during ε -moves. Now we describe a DSA M that performs $2^{c| \langle M \rangle |}$

steps on the empty input. Let $\Gamma = \{0, \dots, n-1\}$. For $i \in 0..n-1$ M verifies if $i \in \mathbb{S}$ and if not M puts i in the set, excludes each $j < i$ and restart the cycle from $i = 0$. It is easy to see that M performs at least 2^n moves since each subset of Γ occurs in the set at some point. Let us describe M more formally.

DSA M is in s_i iff $j \in \mathbb{S}, j < i$; a state q_i corresponds to a program «remove each $j < i$ from \mathbb{S} ». Initially M is in s_0 . On each s_i M tests if $i \in \mathbb{S}$ and moves to s_{i+1} in the case **test+** and moves to q_i otherwise. M moves from q_i to q_{i-1} excluding i on each step until reaches q_0 and then goes to s_0 ; s_n is an accepting state.

Now we prove that there is an NSA recognizing a **NP**-complete problem. This result was also proved independently by M. Kutrib, A. Malcher, M. Wendlandt (private communication with M. Kutrib). We construct a **NP**-complete language that we call **SA-SAT** and reduce **3-SAT** to this language.

Let words $x_i \in \{0, 1\}^*$ encode variables and let a word $\langle \varphi(x_1, \dots, x_n) \rangle$ encode a 3-CNF φ . We define **SA-SAT** to be a language containing the words of the form $x_1 \# x_2 \# \dots \# x_n \# \# \langle \varphi(x_1, \dots, x_n) \rangle$. It is natural to require φ to be satisfiable, but for the needs of set automata we demand more. We build an auxiliary 3-CNF φ' by φ and demand its satisfiability.

We build φ' in the following way. For each variable x_i that appears in the list $x_1 \# x_2 \# \dots \# x_n \# \#$ at least twice, remove all the clauses containing x_i from φ and get the reduced 3-CNF φ'' . We set each variable x of φ'' that is not in the list $x_1 \# x_2 \# \dots \# x_n \# \#$ to zero, simplify φ'' and obtain as a result the 3-CNF φ' . The language **SA-SAT** contains the words $x_1 \# x_2 \# \dots \# x_n \# \# \langle \varphi(x_1, \dots, x_n) \rangle$ such that 3-CNF φ' is satisfiable.

Lemma 2. *The language **SA-SAT** is **NP**-complete and it is recognized by an NSA.*

Proof. Construct NSA M that recognizes the language **SA-SAT**. Firstly, M processes prefix $x_1 \# x_2 \# \dots \# x_n \# \#$, guesses the values b_i of x_i (0 or 1) and put the pairs (x_i, b_i) to the set. If a variable x_i appears more than once in the prefix, then NSA M nondeterministically guesses this event and puts both $(x_i, 0)$ and $(x_i, 1)$ to the set. On processing the suffix $\langle \varphi(x_1, \dots, x_n) \rangle$, for each clause of C NSA M nondeterministically guesses a literal that satisfies the clause and verifies whether the set includes the required value of its variable. It is easy to see that all tests are satisfied iff 3-CNF φ' is satisfiable.

The language **3-SAT** is reduced to **SA-SAT** in a straightforward way. Also it is easy to see that **SA-SAT** \in **NP**.

3 Structural Properties of SA-languages

In this section we show that the class **SA** is a principal rational cone generated by the language of correct protocols. We also prove the lower bounds for the emptiness problem since it directly follows from the fact that **SA** is a cone.

3.1 Protocols

A *protocol* is a word $p = \#u_1\#\text{op}_1\#u_2\#\text{op}_2\#\cdots\#u_n\#\text{op}_n$, where $u_i \in \Gamma^*$, $\# \notin \Gamma$ and $\text{op}_i \in \{\text{in}, \text{out}, \text{test}+, \text{test}-\}$. We say that p is a *correct protocol* for SA M on an input $w \in L(M)$, if M , while processing w , performs the operation op_1 with word u_1 on the work tape at first, then performs op_2 with u_2 on the work tape, and so on until op_n . In the case when M performs a test, op_i is equal to the result of the test: $\text{test}+$ or $\text{test}-$.

We call p a *correct protocol* for SA M , if there exists w such that p is a correct protocol for SA M on input w . And finally, we say that p is a *correct protocol*, if there exists an SA M such that p is a correct protocol for M .

We define SA-PROT to be the language of all correct protocols under the alphabet of the work tape $\Gamma = \{a, b\}$. (By Proposition 3 below each SA-recognizable language is recognized by a SA with the binary alphabet of the work tape.) *Query words* are the words of the form $\#u\#\text{op}$. So a protocol is a concatenation of query words.

Proposition 2. *There exists a DSA M_{PROT} that recognizes the language SA-PROT.*

Proof. Note that the language of all protocols is a regular language. So we assume that the input of M_{PROT} is a protocol. We construct DSA M_{PROT} in a straightforward way. M_{PROT} copies each u_i to the work tape and performs operation op_i . If op_i is a test operation, then M_{PROT} verifies if the result of the test equals to op_i . M_{PROT} accepts a protocol iff all the test results are consistent with the protocol.

Note that the protocol of M_{PROT} on an accepted input w is w by itself. So each protocol accepted by M_{PROT} is a correct protocol by the definition. And it is obvious that M_{PROT} accepts all correct protocols by the construction.

Another observation important for our needs is that for each $L \in \text{SA}$ there exists a SA M with the binary alphabet of the set.

Proposition 3. *Let L be a language recognizable by SA M and $|\Gamma_M| = n$. Then there exists SA M' recognizing L such that $\Gamma_{M'} = \{a, b\}$.*

Proof. Fix some enumeration of letters of Γ_M . SA M' emulates M . When M writes i -th letter on the work tape, M' writes ba^ib . It is clear that for each run of M on w there exists a corresponding run of M' on w such that there is bijection between protocols and all the tests results are the same.

3.2 SA is a Rational Cone

In this subsection we prove that class SA is a principal rational cone generated by the language of correct protocols: $\text{SA} = \mathcal{T}(\text{SA-PROT})$. At first, we give a formal definition of finite state transducer.

Definition 1. *Finite state transducer is defined by a tuple*

$$T = \langle \Sigma, \Gamma, Q, q_0, \delta, F \rangle,$$

where

- Σ is the finite alphabet of the input tape;
- Γ is the finite alphabet of the output tape;
- Q is the finite set of states;
- q_0 is the initial state;
- $F \subseteq Q$ is the set of accepting states;
- $\delta: Q \times (\Sigma \cup \varepsilon) \times Q \times \Gamma^*$ is the transition relation.

We use a notation $q \xrightarrow[v]{u} p$ to express the fact that a transducer T has a run from the state q to the state p such that T on the run processes u on the input tape and writes v on the output tape. The notation is also applied to a single transition. Actually, we have used it in the introduction to define $T(u) = \{v \mid q_0 \xrightarrow[v]{u} q_f, q_f \in F\}$. Recall that $T(L) = \bigcup_{u \in L} T(u)$.

We define $T^{-1}(y) = \{x \mid T(x) = y\}$ and $T^{-1}(A) = \{w \mid T(w) \in A\}$. It is well-known (e.g., see [4]) that for each FST T there exists FST T' such that $T'(u) = T^{-1}(u)$.

We will prove that the class **SA** is a principal rational cone generated by the language **SA-PROT**. It means that for each **SA**-recognizable language L there exists a FST T such that $L = T(\mathbf{SA-PROT})$. Let us describe our plan. At first, we prove that for each **SA** M there is a FST T_M such that $w \in L(M)$ iff $T_M(w) \in \mathbf{SA-PROT}$. We call such FST T_M an *extractor* (of a protocol) for M . Then we show that the required FST T is T_M^{-1} .

Lemma 3. *For each SA $M = \langle S, \Sigma, \Gamma, \triangleleft, \delta_M, s_0, F \rangle$ there exists an extractor $T_M = \langle S \cup \{s'_0\}, \Sigma, \Gamma, \delta, s'_0, F \rangle$.*

Let us informally describe the proof idea. The behavior of **SA** M looks similar to the behavior of the extractor T_M : when FST writes something on the output tape, **SA** writes something on the work tape. So we write $s_i \xrightarrow[y]{x} s_j$ to describe the transition of **SA**: it means that starting from the state s_i on processing $x \in \Sigma \cup \{\varepsilon, \triangleleft\}$ **SA** writes $y \in \Gamma^*$ on the work tape and goes to the state s_j . The only difference is that **SA** knows the results of the performed tests. But nondeterministic extractor can guess the tests results.

Proof. Let us formally define the transition relation δ of T_M . There is an auxiliary transition $s'_0 \xrightarrow[\#]{\varepsilon} s_0$ for the state s'_0 . If M performs no query on a transition $s_i \xrightarrow[y]{x} s_j$, $x \in \Sigma \cup \{\varepsilon\}$, then this transition is included into the transitions of the extractor T_M . If M performs an operation $\text{op} \in \{\mathbf{in}, \mathbf{out}\}$ on the transition $s_i \xrightarrow{x} s_j$, then T_M has the transition $s_i \xrightarrow[\# \text{ op } \#]{x} s_j$; in the case of a test, there

are both transitions for $\text{op} = \mathbf{test}+$ and $\text{op} = \mathbf{test}-$ to the states s_j^+ and s_j^- respectively.

Note that T_M translates each input to a protocol by the construction. If $w \in L(M)$, then each correct protocol for w belongs to T_M : T_M guesses all the M 's tests results. If $w \notin L(M)$, then there is no sequence of test results that allows T_M to write a correct protocol on the output tape.

In the construction above we have a small technical flaw: M can have transitions by the right endmarker \triangleleft . But we do not care about M 's determinism and NSA can nondeterministically guess the right endmarker. The extractor T_M also guesses the last query and doesn't write the last delimiter $\#$.

Theorem 4. *SA is a principal rational cone generated by the language SA-PROT.*

Proof. We shall prove that for each SA M there exists FST T such that $T(\text{SA-PROT}) = L(M)$. Take $T = T_M^{-1}$, where T_M is the extractor for M . By the definition of extractor $w \in L(M)$ iff $T_M(w) \cap \text{SA-PROT} \neq \emptyset$. So if $w \in L(M)$, then there is at least one correct protocol in $T_M(w)$, and we get $w \in T_M^{-1}(\text{SA-PROT})$. In the other direction: if $w \in T_M^{-1}(\text{SA-PROT})$, then $T_M(w) \cap \text{SA-PROT} \neq \emptyset$, and we get that $w \in L(M)$ by the definition of extractor.

3.3 Lower Bounds for the Emptiness Problem

We present an application of the rational cones technique. It is a lower bound on complexity of the emptiness problem for NSA. We show that the emptiness problem for NSA is equivalent to the regular realizability (NRR) problem for SA-PROT. Problem $\text{NRR}(F)$ for language F (a parameter of the problem) is to decide on the input NFA \mathcal{A} whether the intersection $L(\mathcal{A}) \cap F$ is nonempty.

Lemma 4.

$$(L(M) \stackrel{?}{=} \emptyset) \leq_{\log} \text{NRR}(\text{SA-PROT}) \text{ and } \text{NRR}(\text{SA-PROT}) \leq_{\log} (L(M) \stackrel{?}{=} \emptyset).$$

Proof. It is easy to see that one can construct the extractor T_M by SA M in log space. So, by the definition of extractor, we get that $L(M) = \emptyset$ iff $T_M(\Sigma^*) \cap \text{SA-PROT} = \emptyset$. Note that rational transductions preserve regularity [4]: $T_M(\Sigma^*) \in \text{REG}$. It is shown in [6] that NFA recognizing $T_M(\Sigma^*)$ is log-space constructible by the description of T_M . So $(L(M) \stackrel{?}{=} \emptyset) \leq_{\log} \text{NRR}(\text{SA-PROT})$.

Now consider a regular language $R \subseteq \Gamma^*$. Note that $R \cap \text{SA-PROT} = \emptyset$ iff $L(M_{\text{PROT}}) \cap R = \emptyset$. It was shown in [3] that there exists SA M_R recognizing $L(M_{\text{PROT}}) \cap R$. This SA is also constructible in log space by the description of NFA recognizing R : the proof is almost the same as for the aforementioned NFA by FST construction. Thus $\text{NRR}(\text{SA-PROT}) \leq_{\log} (L(M) \stackrel{?}{=} \emptyset)$.

Also we need the following relation between RR-problems and rational transductions.

Proposition 4 ([6]). *If $A \leq_{\text{rat}} B$, then $\text{NRR}(A) \leq_{\log} \text{NRR}(B)$.*

We use the hardness of NRR problem for languages Per_k from Example 1.

Theorem 5 ([7], [8]). *The problem $\text{NRR}(\text{Per}_1)$ is **NP**-complete and the problem $\text{NRR}(\text{Per}_2)$ is **PSPACE**-complete.*

From these facts we derive the **PSPACE** lower bound.

Theorem 6. *For SA the emptiness problem is **PSPACE**-hard. For SA with the unary alphabet of the work tape the emptiness problem is **NP**-hard.*

Proof. At first, we show that the emptiness problem is **PSPACE**-hard for NSA. By Proposition 3 we assume that SA has the binary alphabet of the work tape. The language Per_2 is recognizable by DSA (see Example 1). By Theorem 4 we get that $\text{Per}_2 \leq_{\text{rat}} \text{SA-SAT}$. By Proposition 4 we obtain $\text{NRR}(\text{Per}_2) \leq_{\log} \text{NRR}(\text{SA-SAT})$ and therefore problem $\text{NRR}(\text{SA-SAT})$ is **PSPACE**-hard. So, by Lemma 4, the emptiness problem for NSA is **PSPACE**-hard too.

To prove that the emptiness problem is hard for DSA we reduce the problem for NSA to the problem for DSA. It is easy to build by NSA M a DSA M' such that $L(M) = \emptyset$ iff $L(M') = \emptyset$. Assume that $\Sigma_{M'} = \Sigma_M \cup \{\#, \$\}$, $\#, \$ \notin \Sigma_M$ and M has n nondeterministic transitions in $\delta(s, a)$. We replace i -th transition $\delta(s, a) \vdash (s', x)$ by the chain of deterministic transitions by the word $\#\$^i\#a$: $\delta(s, \#) = (s_\#, \varepsilon)$, $\delta(s_\#, \$) = (s_{\#\$}, \varepsilon)$, \dots , $\delta(s_{\#\$^k}, \$) = (s_{\#\$^{k+1}}, \varepsilon)$, \dots , $\delta(s_{\#\$^i}, \#) = (s_{\#\$^i\#}, \varepsilon)$, $\delta(s_{\#\$^i\#}, a) = (s', x)$. If $w \in L(M)$, then by the construction of M' there exists a word $w' \in L(M')$. If M' accepts a word w' , then M accepts a word w which is obtained from w' by erasing all the symbols $\#$ and $\$$.

The case of SA with the unary stack alphabet is similar. We only use language SA-PROT_1 (with $u_i \in a^*$) of protocols for SA with the unary alphabet of the work tape.

4 SA \subseteq NP

We come to the main result of the paper. The general idea is to prove that for each $w \in L(M)$ there exists a short (polynomial in $|w|$) protocol p for w . Thus an NP-algorithm guesses the run of M on w corresponding to p and verifies that M accepts w on that run.

We start with auxiliary notions.

Consider a protocol $p = \#u_1\#\text{op}_1\#u_2\#\text{op}_2\#\dots\#u_n\#\text{op}_n$. By a *segment* p_i of p we mean an occurrence of a query word $\#u_i\#\text{op}_i$ in the protocol. The different segments may coincide as words. We say that a segment p_i *supports* segment p_j if $u_i = u_j$ and $\text{op}_i = \mathbf{in}$, $\text{op}_j = \mathbf{test+}$ or $\text{op}_i = \mathbf{out}$, $\text{op}_j = \mathbf{test-}$ and there is no segment p_k such that $\text{op}_k \in \{\mathbf{in}, \mathbf{out}\}$, $u_k = u_i$ and $i < k < j$.

We say that segments v_1, v_2, \dots, v_k form a *chain* C if v_1 supports v_i , $i \in 2..k$. *Standalone queries* are the segments $\#u\#\mathbf{in}$ and $\#u\#\mathbf{out}$ that support no segments and the segments $\#u\#\mathbf{test-}$ having no support. Each standalone query forms a *standalone chain*. Let $v_i = \#u\#\text{op}_i$ be the segments of a chain. We denote the chain by $C(u)$ and call u as the *pivot* of $C(u)$. Also we denote the chain C^+ if $\text{op}_1 = \mathbf{in}$ and C^- if $\text{op}_1 \in \{\mathbf{out}, \mathbf{test-}\}$.

The following lemma directly follows from the definitions.

Lemma 5. *A protocol p is correct iff each segment $\#u_i\#\mathbf{test}+$ is supported and each segment $\#u_i\#\mathbf{test}-$ is either supported, or standalone.*

Let us give a formal definition of SA's run. Consider the process of SA performing query. Let M starts from a state s (with the blank work tape), writes u on the work tape on processing x on the input tape and performs a query at a state q . We denote this sequence of operations as $s \xrightarrow[u]{x} q$. Then the *accepting run* of SA M on word $w = x_0x_1 \cdots x_{n-1}, x_i \in \Sigma^*$ is a sequence of queries

$$s_0 \xrightarrow[u_0]{x_0} s_1 \xrightarrow[u_1]{x_1} s_2 \xrightarrow[u_2]{x_2} \cdots \xrightarrow[u_{n-1}]{x_{n-1}} s_n, \quad s_n \in F. \quad (1)$$

We fix the states s_0, s_1, \dots, s_{n-1} and the states that arises on writing the words u_i (we skip them in (1)). We denote $s_i \xrightarrow[u_i]{x_i} s_{i+1} \xrightarrow[u_{i+1}]{x_{i+1}} \cdots \xrightarrow[u_{j-1}]{x_{j-1}} s_j$ as $s_i \xrightarrow{x_i x_{i+1} \cdots x_j} s_j$.

Note that some x_i may be ε and it is the main obstacle to prove an existence of a short protocol: on a segment $s_i \xrightarrow{\varepsilon} s_j$ a lot of queries, that supports the tests on the nonempty x_k , may be performed. So the total number of queries may be super-polynomial.

To overcome this difficulty we will use a special form of NSA that we call *Atomic Action Normal Form* (AANF).

4.1 Atomic Action Normal Form

AANF is a refinement of the infinite action normal form from [3] developed for DSA.

We will also use the notation $s \xrightarrow[u]{x} q$ for a relation that holds if M can perform a query on u after processing x starting from s and querying at q . We always indicate the meaning of the notation: a segment or a relation. A relation $s \xrightarrow[u]{X} q$ means that $U = \{u \mid \exists x \in X : s \xrightarrow[u]{x} q\}$.

At first, we provide the definition of the action normal form (ANF) and then we define AANF as a special form of ANF.

Definition 2. *We say that SA M is in the action normal form if the initial state appears only once on each computational path of M , and each state is either marked by operations **test**+, **test**-, **in**, **out**, if on a transition to this state M performs the respective operation, or marked by **write** in the other case (we assume that if SA doesn't perform a query, then it writes something on the work tape, maybe ε).*

Lemma 6 ([3]). *For each SA M there exists SA M' in ANF such that $L(M) = L(M')$.*

So, if M is in ANF, then $S = \{s_0\} \cup S_{\mathbf{test}+} \cup S_{\mathbf{test}-} \cup S_{\mathbf{in}} \cup S_{\mathbf{out}} \cup S_{\mathbf{write}}$, $S_{\mathbf{op}} \cap S_{\mathbf{op}'} = \emptyset$, $\mathbf{op} \neq \mathbf{op}'$. Each state q such that relation $s \xrightarrow[u]{x} q$ holds is marked by the operation of the query; states that occur while writing u are marked as

write. Denote $S_{\text{begin}} = \{s_0\} \cup S_{\text{test}+} \cup S_{\text{test}-} \cup S_{\text{in}} \cup S_{\text{out}}$, $S_{\text{end}} = S_{\text{begin}} \setminus \{s_0\}$ the sets of the start and the end states of queries. We denote L_{s_i, s_j} a language such that relation $s_i \xrightarrow[\text{ }]{\Sigma^*} s_j$ holds, $s_i \in S_{\text{begin}}$, $s_j \in S_{\text{end}}$. So, L_{s_i, s_j} consists of all the words that M can write on the work tape on a path from s_i to s_j on processing some word without performing queries in between.

In fact Lemma 6 was stated for DSA: the proof doesn't depends on determinism, but it preserve determinism of SA.

We restate Lemma 8 from [3] in a way convenient for our needs.

Lemma 7 (Lemma 8 [3]). *Let F be a finite language. For SA M in ANF there exists SA M' in ANF such that for each pair of states $s'_i \in S'_{\text{begin}}$, $s'_j \in S'_{\text{end}}$ $L_{s'_i, s'_j} = L_{s_i, s_j} \setminus F$ for some s_i, s_j .*

This lemma implies that each SA can be converted to SA in *infinite action normal form*: an ANF such that each language L_{s_i, s_j} is infinite. But for our purposes we need more strict conditions on L_{s_i, s_j} .

Definition 3. *We say that NSA M is in AANF if M is in ANF and the following conditions hold.*

- *An each state is marked as a state that either can occur only on ε -segment $s \xrightarrow[u]{\varepsilon} q$ or only on non- ε segment $s' \xrightarrow[u]{x} q'$, $x \neq \varepsilon$; state q' may be marked as ε -state as the end of a segment (because there could be segments $s' \xrightarrow[u]{x} q' \xrightarrow[u']{\varepsilon} q''$). The same is valid for the end of the ε -segment: it may be marked as non- ε .*
- *There exists a finite family of regular languages \mathcal{L}_M such that*
 - *for all $A_\alpha, A_\beta \in \mathcal{L}_M$: $A_\alpha \cap A_\beta = \emptyset$ if $\alpha \neq \beta$;*
 - *if $s \xrightarrow[U]{\Sigma^*} q$, $s' \xrightarrow[U']{\Sigma^*} q'$, then either $U \cap U' = \emptyset$, or $U = U' \in \mathcal{L}_M$;*
 - *for each $A_\alpha \in \mathcal{L}_M$: $|A_\alpha| = \infty$.*

Informally, AANF implies that each $s \xrightarrow[u]{x} q$ belongs to an equivalence class corresponding to $A_\alpha \in \mathcal{L}_M$, $u \in A_\alpha$, and the number of such classes is finite. Moreover, in the case of $x = \varepsilon$ the relation $s \xrightarrow[A_\alpha]{\varepsilon} q$ holds, and it means that one can replace u (on the segment) by any word $u_\alpha \in A_\alpha$ and obtain a run for w again. But this run may be incorrect, so later we will carefully choose u_α to maintain the correctness of the run. Thus, AANF helps us to struggle ε -queries as well as to make non- ε -segments short.

Lemma 8. *For each NSA M there exists NSA M' in AANF such that $L(M) = L(M')$.*

Proof. Due to Lemma 6 we assume that NSA M is in ANF. We construct an auxiliary SA \tilde{M} first. We replace each state from $s_i \in S_{\text{begin}}$ by states $(s_i, \varepsilon), (s_i, \neq)$, so if M has a transition to s_i , then \tilde{M} has corresponding transitions to the both states $(s_i, \varepsilon), (s_i, \neq)$. We require \tilde{M} to process only the empty word during each

query that starts in (s_i, ε) and process at least one letter when starts in (s_i, \neq) . It is easy to implement such constraints since they are the regular constraints on the input tape. So we just also mark **write** states by ε and \neq and in the first case just throw out all \neq -moves and in the second case use extra bit to verify if at least one letter have been processed before performing the query. Such transformation leaves \tilde{M} in ANF.

Let $\mathcal{L} = \{L_{s_i, s_j} \mid s_i \in S_{\text{begin}}, s_j \in S_{\text{end}} \text{ states of } \tilde{M}\}$. It is a finite family since the number of \tilde{M} 's states is finite. At first we construct a family $\mathcal{L}_{M'}$ by \mathcal{L} .

We take the closure of \mathcal{L} under \cap and \setminus operations and obtain a finite family \mathcal{L}'' . Next, we remove from \mathcal{L}'' all the languages that are the unions of some other languages and obtain a finite family \mathcal{L}' . By the construction, each language from \mathcal{L} is a finite union of some languages of \mathcal{L}' , but some languages from \mathcal{L}' can be finite. Let F be a union of all finite languages from \mathcal{L}'_M and finally let $\mathcal{L}_{M'}$ consists of all infinite languages from \mathcal{L}' .

Now we show how to transform \tilde{M} to M' in such a way that $\mathcal{L}_{M'} = \{L_{s'_i, s'_j} \mid s'_i \in S'_{\text{begin}}, s'_j \in S'_{\text{end}} \text{ states of } M'\}$. Note that $\mathcal{L}_{M'}$ is a family of regular languages and so is \mathcal{L} . Denote $\mathcal{L}_{M'} = \{A_1, \dots, A_\alpha, \dots, A_m\}$. At first, we transform M to M'' by applying Lemma 7 for M and F . We get that $L_{s''_i, s''_j} = \bigcup_{\alpha \in I} A_\alpha$, where $I \subseteq \{1, \dots, m\}$.

Now we construct M' by M'' . We replace a state s''_i by states (s''_i, α) for each $\alpha \in I$ and demand that M' , starting from (s''_i, α) , writes on the work tape only the words from A_α . We implement such constraints in the same way as at the begin of the proof.

So, $\mathcal{L}_{M'} = \{L_{s'_i, s'_j} \mid s'_i \in S'_{\text{begin}}, s'_j \in S'_{\text{end}} \text{ states of } M'\}$ and all the constraints of AANF on A_α s are satisfied by the construction and all the states are marked according to the definition of AANF.

4.2 The main part of the proof

From now on we assume that M is in AANF. We say that p_i is an ε -segment of a protocol if $s_i \xrightarrow[u_i]{\varepsilon} s_{i+1}$. We say that an ε -segment p_i has type α if the relation $s_i \xrightarrow[A_\alpha]{\varepsilon} s_{i+1}$ holds for $A_\alpha \in \mathcal{L}_M$. We call a chain $C(u)$ of ε -segments an ε -chain. We denote ε -chain C_α if all the segments have type α . Different segments of an ε -chain can't have different types due to AANF construction.

Now we are ready to describe the main steps of the proof. Each step is a transformation of a run (1) to some other correct run of M on w . We call such transformations *valid*. We call a word u *short* if $|u|$ is polynomial in $|w|$. We call a run *short* if the corresponding protocol is short.

Lemma 9. *For each $A_\alpha \in \mathcal{L}_M$ there exist short words $u_\alpha^+ \in A_\alpha$ and $u_\alpha^- \in A_\alpha$ such that the simultaneous replacement of the pivots in all segments of ε -chains C_α^+ and C_α^- by u_α^+ and u_α^- respectively is a valid transformation of a run. Moreover*

- M never adds each word u_α^- to \mathbb{S} ;
- M never removes each word u_α^+ from \mathbb{S} ;
- all segments of ε -chains C^- become standalone;
- the length of each word u_α^\pm is $O(|w|)$.

Proof. As we have already mentioned in the proof of Theorem 7, there is no more than $|w|$ nonempty x_i s and therefore no more than $|w|$ pivots u_i of non- ε -chains.

Since each $A_\alpha \in \mathcal{L}_M$ is an infinite regular language it contains a sequence of words, the lengths of which form an arithmetic progression. We choose words $u_\alpha^+ \neq u_\alpha^-$ from this sequence such that both of them differs from any pivot u_i of a non- ε -chain.

Perform the replacement described in the lemma. One can replace u_i by u_α^\pm on each ε -segment due to AANF construction. So to prove that such a transformation is valid we shall prove that the protocol transfers to a correct one. We prove it using Lemma 5: show that each query is either supported, or a standalone one.

Note that, by the choice of u_α^\pm , SA M never adds a word u_α^- to \mathbb{S} and never removes a word u_α^+ from \mathbb{S} on the resulting run. Therefore all the queries of ε -chains C_α^- become isolated: **test**-queries by the words that never occur in \mathbb{S} requires no support and **out**-queries do not support any **test**-query anymore. All the queries of each ε -chain C_α^+ are remain supported since u_α^+ never removes from the set after the first addition, and each test of each ε -chain C_α^+ tests if $u_\alpha^+ \in \mathbb{S}$.

Since u_α^\pm do not affect neither non- ε -segments, nor ε -queries that are the pivots of non- ε -chains, each non- ε -query remains either supported or isolated, as it was before the transformation.

From now on we assume that run (1) has all the properties described in Lemma 9. To prove the main result we need two following lemmata.

Lemma 10. *If for each segment $s_i \xrightarrow[u_i]{x_i} s_{i+1}$, $x_i \neq \varepsilon$, of a run (1) the word u_i is short, then there is a valid transformation of a run (1) to a short run.*

Proof. Let $\mathcal{L}_M = \{A_1, A_2, \dots, A_m\}$. Recall that m is a constant depending on the description of SA M but not w . We apply Lemma 9 to the run first. Denote U the set of words u_α^\pm . We mark each state s_i of the run (1) by a vector $\mathbf{a} = (a_1, \dots, a_m)$ depending on the content of \mathbb{S} at this point of the run. A component a_α consists of two bits: the first one marks whether \mathbb{S} contains a word from $A_\alpha \setminus U$ and the second marks whether $u_\alpha^+ \in \mathbb{S}$:

- $a_\alpha = 00$ if $u_\alpha^+ \notin \mathbb{S}$ and $\mathbb{S} \setminus U \cap A_\alpha = \emptyset$;
- $a_\alpha = 01$ if $u_\alpha^+ \in \mathbb{S}$ and $\mathbb{S} \setminus U \cap A_\alpha = \emptyset$;
- $a_\alpha = 10$ if $u_\alpha^+ \notin \mathbb{S}$ and $\mathbb{S} \setminus U \cap A_\alpha \neq \emptyset$;
- $a_\alpha = 11$ if $u_\alpha^+ \in \mathbb{S}$ and $\mathbb{S} \setminus U \cap A_\alpha \neq \emptyset$.

Consider an ε -segment of the run $s_i \xrightarrow{\varepsilon} s_j$. Recall that

$$s_i \xrightarrow{\varepsilon} s_j = s_i \xrightarrow[u_i]{\varepsilon} s_{i+1} \xrightarrow[u_{i+1}]{\varepsilon} \dots \xrightarrow[u_{j-1}]{\varepsilon} s_j.$$

We fix the partition of the segment $s_i \rightarrow s_j$

$$s_i \xrightarrow{\varepsilon} s_{l_1} \xrightarrow[u_{l_1}]{\varepsilon} s_{r_1} \xrightarrow{\varepsilon} s_{l_2} \xrightarrow[u_{l_2}]{\varepsilon} s_{r_2} \xrightarrow{\varepsilon} \dots s_{l_k} \xrightarrow[u_{r_k}]{\varepsilon} s_{r_k} \xrightarrow{\varepsilon} s_j,$$

such that $s_{l_k} \xrightarrow[u_{l_k}]{\varepsilon} s_{r_k}$ are all the queries of the segment that supports non- ε -chains; $s_i = s_{r_0}$, $s_j = s_{l_{k+1}}$.

Let \mathbb{S}_k denotes the set content when M is in the state s_k and \mathbf{a}^k is the corresponding vector. Due to Lemma 9 M can perform on a segment $s_{r_k} \xrightarrow{\varepsilon} s_{l_{k+1}}$ **in**, **out** and **test**- queries with only the words from U on the work tape. So, the first bit of a_α^k never changes on the segment $s_{r_k} \xrightarrow{\varepsilon} s_{l_{k+1}}$. But if M performs a **test**+ query $s_n \xrightarrow[u_n]{\varepsilon} s_{n+1}$, $u_n \notin U$, $r_k \leq n \leq l_{k+1}$ and s_n has type α then $a_\alpha^k \in \{11, 10\}$. That's why we need the first bit of each component of the vector \mathbf{a}^k .

Note that each query $s_{l_k} \xrightarrow[u_{l_k}]{\varepsilon} s_{r_k}$ is short due to the lemma's condition and the number of all such segments doesn't exceed $|w|$. So, our goal is to show that there is a valid transformation of the run such that the segments $s_{r_k} \xrightarrow{\varepsilon} s_{l_{k+1}}$ are turned to the short ones: the segments $s_{l_k} \xrightarrow{\varepsilon} s_{r_k}$ and the non- ε -segments are already short.

For this purpose we consider the extractor T_M . By the construction of the extractor in Lemma 3 there is a bijection between the states of the SA and the subset of extractor states such that segments of SA's run $s \xrightarrow{\varepsilon} q$ corresponds to the segments of the extractor's run. So, we consider the segments of the corresponding extractor's run $s_{r_k} \xrightarrow[p_{r_k} p_{r_k+1} \dots p_{l_{k+1}}]{\varepsilon} s_{l_{k+1}}$. Let R_k be the language

such that the relation $s_{r_k} \xrightarrow[R_k]{\varepsilon} s_{l_{k+1}}$ holds.

Let NFA \mathcal{A} recognizes the language R_k . Now we construct an auxiliary graph G by the graph of \mathcal{A} . The vertex set of G is the set $Q \times \{00, 01, 10, 11\}^m = V$, where $Q \subseteq S_{\text{begin}} \cup S_{\text{end}}$ is the set of \mathcal{A} 's states. We mark the states of \mathcal{A} in the same manner that we have marked the states of the run. So we add edges to G in the following way.

Let the relation $q_l \xrightarrow[A_\alpha]{\varepsilon} q_r$ hold for the states q_l, q_r . Then there is an edge between (q_l, \mathbf{a}) and (q_r, \mathbf{a}') iff

- $q_r \in S_{\text{out}} \cup S_{\text{test-}}$ and $\mathbf{a} = \mathbf{a}'$;
- $q_r \in S_{\text{test+}}$ and $\mathbf{a} = \mathbf{a}'$, $a_\alpha \neq 00$;
- $q_r \in S_{\text{in}}$ and $\mathbf{a}_\beta = \mathbf{a}'_\beta$, $\beta \neq \alpha$, and $\mathbf{a}'_\alpha = 01$ if $\mathbf{a}_\alpha \in \{00, 01\}$; $\mathbf{a}'_\alpha = 11$ if $\mathbf{a}_\alpha \in \{10, 11\}$.

NFA \mathcal{A} is easy constructible by T_M since M is in AANF: \mathcal{A} 's graph is a subgraph of T_M that consists of all the states marked by ε . So the number of such automata depends only on M 's description as well as the number of corresponding graphs. So the size of G is $O(1)$ by $|w|$. Now return to the segment $s_{r_k} \xrightarrow{\varepsilon} s_{l_{k+1}}$ of the SA's run. Let \mathbf{a}^{r_k} and $\mathbf{a}^{l_{k+1}}$ be the corresponding vectors

for $s_{r_k}, s_{l_{k+1}}$. We take the short path in G from $(s_{r_k}, \mathbf{a}^{r_k})$ to $(s_{l_{k+1}}, \mathbf{a}^{l_{k+1}})$. This path has a constant length since G has size $O(1)$ by $|w|$. By this path we recover a new segment $p'_{r_k, l_{k+1}} \in R_k$ which is short since it consists of constant number of queries and each query is either a query by u_α^\pm or by a short word u_i from a non- ε -chain. Due to the construction of vectors \mathbf{a}^k , the replacement of protocol segments $p_{r_k} \cdot p_{r_k+1} \cdots p_{l_{k+1}}$ by $p'_{r_k, l_{k+1}}$ doesn't change the set $\mathbb{S}_{l_{k+1}}$. So the replacement of all the segments $s_{r_k} \xrightarrow{\varepsilon} s_{l_{k+1}}$ of the run by the constructed segments is a valid transformation of the run.

Lemma 11. *Assume that $C(u)$ is a non- ε -chain of segments $v_1, \dots, v_k, v_i = \#u\# \text{op}_i$. Then there is a short word u' such that replacement of v_i by $\#u'\# \text{op}_i$ is a valid transformation of run (1).*

Proof. At first, we apply Lemma 9 to the run. Consider non- ε -chain $C(u)$. It consists of the run's segments $s_i \xrightarrow[u]{x_i} s_{i+1}$ and some of these segments are ε -segments. Due to the definition of AANF the relation $s_i \xrightarrow[A_\alpha]{\Sigma^*} s_{i+1}$ holds for some A_α and all the segments of C . At the same time the relation $s_i \xrightarrow[A_\alpha]{\varepsilon} s_{i+1}$ holds for each ε -segment of C . So if we find a short word u' such that for each non- ε -segment of C the relation $s_i \xrightarrow[u']{x_i} s_{i+1}$ holds, then the relation $s_i \xrightarrow[u']{\varepsilon} s_{i+1}$ holds for all ε -segments of the chain.

Denote by R_i a regular language such that the relation $s_i \xrightarrow[R_i]{x_i} s_{i+1}$ holds, $x_i \neq \varepsilon$. We will prove that there is a short word $u' \in \bigcap R_i \setminus F$, where F is a finite set of forbidden words. Recall that the replacement of u by u' should not break any other chain. Thus, we take F consisting of u_α^\pm and all the words u_i (except of u) such that there is a chain $C(u_i)$ in the run (1). Note that $|F|$ is $O(|w|)$ since the number of u_α^\pm is $O(1)$ by Lemma 9 and the number of all non- ε -chains is $O(|w|)$.

Note that the number of R_i is bounded by $|w|$, but it may grow with $|w|$. So if R_i were arbitrary regular languages, the intersection $\bigcap R_i$ could contain no short words.

To overcome this difficulty we exploit a specific form of languages R_i . Let $x_i = w_{l_i} w_{l_i+1} \cdots w_{r_i}, w_k \in \Sigma$. Let q_k be the states such that $q_k \xrightarrow{w_k} q_{k+1}$ during the run (1); if $k = l_i$, then $q_k = s_i$; if $k+1 = r_i$, then $q_{k+1} = s_{i+1}$. There could be many choices for q_k inside the segment $s_i \hookrightarrow s_{i+1}$ – there are no restrictions on that choice. Define $R(w_k)$ to be a language such that the relation $q_k \xrightarrow[R(w_k)]{w_k} q_{k+1}$ determined by the extractor T_M holds. As in the proof of Lemma 10, all such languages are regular and their number depends only on the description of M , but not w . So, we get that $R_i = R(w_{l_i}) \cdot R(w_{l_i+1}) \cdots R(w_{r_i}) = R_1^i \cdot R_2^i \cdots R_{m_i}^i$, where $R_j^i = R(w_{l_i+j-1})$, $m_i = r_i - l_i + 1$.

Since the number of $x_i \neq \varepsilon$ does not exceed $|w|$, we get that $\sum_i m_i \leq |w|$. Note that $u \in \bigcap_{i=1}^m R_1^i \cdot R_2^i \cdots R_{m_i}^i \setminus F$ by the definition of R_j^i . It implies that for

each i there is a factorization of $u = v_1 \cdot v_2 \cdots v_{m_i}$ such that $v_j \in R_j^i$. So, there exists a partition $u = \tilde{u}_1 \tilde{u}_2 \cdots \tilde{u}_K, \tilde{u}_k \in \Gamma^*$ such that all factors of the above factorizations have the form $\tilde{u}_l \tilde{u}_{l+1} \cdots \tilde{u}_r \in R_j^i$. It is easy to see that the length K of the partition satisfies inequality $K \leq \sum_i m_i \leq |w|$.

Let NFA \mathcal{A}_j^i recognizes R_j^i . For each \mathcal{A}_j^i there exists a successive run

$$q_l \xrightarrow{\tilde{u}_l} q_{l+1} \xrightarrow{\tilde{u}_{l+1}} q_{l+2} \rightarrow \cdots \rightarrow q_k \xrightarrow{\tilde{u}_k} q_{k+1} \rightarrow \cdots \xrightarrow{\tilde{u}_r} q_{r+1},$$

where q_l is the initial state and q_{r+1} is an accepting state of \mathcal{A}_j^i . Define NFA \mathcal{B}_k^i as a modification of \mathcal{A}_j^i by replacement of the initial state to q_k and the set of accepting states to $\{q_{k+1}\}$. Then $L(\mathcal{A}_j^i) \supseteq L(\mathcal{B}_l^i) \cdot L(\mathcal{B}_{l+1}^i) \cdots L(\mathcal{B}_r^i)$ and

$$R_i \supseteq L(\mathcal{B}_1^i) \cdot L(\mathcal{B}_2^i) \cdots L(\mathcal{B}_K^i).$$

Note that for each j the intersection $\bigcap_i L(\mathcal{B}_j^i) = R'_j$ is nonempty and contains \tilde{u}_j .

We prove that required u' exists and belongs to $R'_1 \cdot R'_2 \cdots R'_K$. If so, then u' evidently belongs to $\bigcap_i R_i$, since $\bigcap_i R_i \supseteq R'_1 \cdot R'_2 \cdots R'_K$.

As was already mentioned for other objects, the number of all possible \mathcal{B}_j^i is finite and depends on M but not w . The same is true for the state complexities of R'_j . If all the R'_j are finite languages, then the length of the longest word from R'_j is $O(1)$ by $|w|$ and therefore $u' = u$ is a short word: it belongs to $R'_1 \cdot R'_2 \cdots R'_K$ by the construction. If at least one language R'_j is infinite, then, as in the proof of Lemma 9, it contains a sequence of words with linear length's growth. Fix all the shortest words $u'_k \in R'_k$, $k \neq j$, and choose the first word $u'_j \in R'_j$ from this sequence such that $u' = u'_1 \cdot u'_2 \cdots u'_j \cdots u'_K \notin F$. The word u' is short since $|F|$ is $O(|w|)$.

Tying up loose ends we get the main result.

Theorem 7. *There is an NP-algorithm verifying whether $w \in L(M)$: $\text{SA} \subseteq \text{NP}$.*

Proof. We assume that M is an NSA in AANF by Lemma 8. By definition, $w \in L(M)$ iff there exists an accepting run (1). We show that in this case there is also a short run. Thus, an NP-algorithm guesses a short run and checks its correctness.

At first, we apply Lemma 9 to get a run satisfying the conditions of the lemma. After that we shall turn all non- ε -segments to short ones. Note that there is no more than $|w|$ non- ε -segments in any run and therefore there is no more than $|w|$ non- ε -chains. We apply Lemma 11 no more than $|w|$ times to get a run in which all segments $s_i \xrightarrow[u_i]{x_i} s_{i+1}$, $x_i \neq \varepsilon$, are short. Finally, we apply Lemma 10 and get the short run for w .

References

1. Kutrib, M., Malcher, A., Wendlandt, M.: Deterministic set automata. In Shur, A.M., Volkov, M.V., eds.: *Developments in Language Theory: 18th International Conference, DLT 2014, Ekaterinburg, Russia, August 26-29, 2014. Proceedings.* Springer International Publishing, Cham (2014) 303–314
2. Kutrib, M., Malcher, A., Wendlandt, M.: Regularity and size of set automata. In Jürgensen, H., Karhumäki, J., Okhotin, A., eds.: *Descriptive Complexity of Formal Systems: 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings.* Springer International Publishing, Cham (2014) 282–293
3. Kutrib, M., Malcher, A., Wendlandt, M.: Set automata. *International Journal of Foundations of Computer Science* **27**(02) (2016) 187–214
4. Berstel, J.: *Transductions and context-free languages.* Ed. Teubner (1979)
5. Ladner, R.E.: The circuit value problem is log space complete for p. *SIGACT News* **7**(1) (January 1975) 18–20
6. Rubtsov, A.A., Vyalyi, M.N.: Regular realizability problems and context-free languages. In Shallit, J., Okhotin, A., eds.: *Descriptive Complexity of Formal Systems.* Volume 9118 of *Lecture Notes in Computer Science.* Springer International Publishing (2015) 256–267
7. Anderson, T., Loftus, J., Rampersad, N., Santean, N., Shallit, J.: Special issue: 2nd international conference on language and automata theory and applications (lata 2008) detecting palindromes, patterns and borders in regular languages. *Information and Computation* **207**(11) (2009) 1096 – 1118
8. Vyalyi, M.N.: On the models of nondeterminism for two-way automata (in russian). *Proceedings of VIII international conference «Discrete models in the theory of control systems».* (2009) 54–60